Multimedia FAQ

发行版本 (HEAD detached at 9f3e22c)

SOPHGO

2024 年 07 月 25 日

目录

发布记录

Copyright © SOPHGO

CHAPTER 1

声明

法律声明

版权所有 © 算能 2022. 保留一切权利。
非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

注意

您购买的产品、服务或特性等应受算能商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，算能对本文档内容不做任何明示或默示的声明或保证。由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

技术支持

地址
北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

邮编
100094

网址
https://www.sophgo.com/

邮箱
sales@sophgo.com

电话
+86-10-57590723 +86-10-57590724

SDK 发布记录

版本	发布日期	说明
v22.09.02	2022.09.02	第一次发布。
v22.10.01	2022.10.01	第二次发布。
v22.12.01	2022.12.01	2022 年 12 月第一次发布。
v23.03.01	2023.03.01	2023 年 3 月第一次发布。
v23.05.01	2023.05.01	2023 年 5 月第一次发布。
v23.07.01	2023.07.01	2023 年 7 月第一次发布。
v23.10.01	2023.10.01	2023 年 10 月第一次发布。

Copyright © SOPHGO

CHAPTER 2

多媒体客户常见问题列表

2.1 4K 图片的问题

答：有些客户需要的图片较大如 8K 等，由于 VPP 只支持 4K 大小的图片，所以通过 opencv 读取图片后，会自动保持比例缩放到一个 4K 以内的尺寸。

如果客户需要传递原始图片的坐标位置给远端，可以有以下两种做法：

1. imread 中设置 flag = IMREAD_UNCHANGED_SCALE，此时图片不会真正解码，会在 mat.rows/cols 中返回图片的原始宽高，然后可根据缩放比例计算得到原图的坐标
2. 传递相对坐标给远端，即坐标 x/缩放后的宽，坐标 y/缩放后的高传递到远端。这步相对坐标计算也可以在远端完成，然后可以根据远端知道的原始图像宽高计算得到正确的原图坐标

2.2 Opencv 读取图片后，cvMat 转为 bmimage，之后，调用 bmcv_image_vpp_convert 做缩放或者颜色空间转换，得到的图片不一致

原因分析：opencv 内部的转换矩阵和 bmcv_image_vpp_convert 使用的转换矩阵不一致，需要调用 bmcv_image_vpp_csc_matrix_covert，并且指定 CSC_YPbPr2RGB_BT601 来进行转换才能保持一致。

2.3 Opencv imread 读取图片性能问题。

原因分析：如果碰到图片小于 16x16 大小的图片，或者 progressive 格式的 jpeg，芯片不能实现加速，结果走了 CPU 的路径，导致客户发现图片解码并没有加速。

2.4 VideoWriter.write 性能问题，有些客户反应，存文件慢。

解析：就目前来看看采用 YUV 采集，然后编码 10-20ms 之间写入一帧数据属于正常现象。

2.5 Ffmpeg 的阻塞问题

原因分析：如果没有及时释放 avframe，就会导致阻塞，vpu 内部是循环 buffer。

2.6 关于什么时候调用 uploadMat/downloadMat 接口的问题。

解析：当创建了一个 cv::Mat, 然后调用 cv::bmcv 里面的接口进行了一些处理后，设备内存的内容改变了，这时候需要 downloadMat 来同步一下设备内存和系统内存。当调用了 cv::resize 等 opencv 原生的接口后，系统内存的内容改变了，需要 uploadMat，使设备内存和系统内存同步。

2.7 opencv 下如何获取视频帧的 timestamp ?

答：opencv 原生提供了获取 timestamp 的接口，可以在 cap.read() 每一帧后获取当前帧的 timestamp.

代码如下：

```
Mat frame;
cap.read(frame);
/* 获取timestamp，返回值为double类型 */
int64_t timestamp = (int64_t)cap.getProperty(CAP_PROP_TIMESTAMP);
```

2.8 SA3 opencv 下 videocapture 经常 5 分钟左右断网的解决方案

答：在 udp 方式下,SA3 经常发生 RTSP 数据连上后 3-5 分钟就” connection timeout” 的问题，这个问题最终解决方案是更新最新的路由板软件。验证方法可以用 TCP 测试下，如果 TCP 没有问题可以确认是这类问题。

使用 TCP 的方式见下面：

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;tcp"
```

执行应用（如果用 sudo 执行，需要 sudo -E 把环境变量带过去）注意：最新的 middleware-soc 将使用 TCP 作为默认协议，对原来客户需要使用 UDP 传输协议的，需要引导客户按照下面方式进行设置

使用 UDP 方式：

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;udp"
```

执行应用（如果用 sudo 执行，需要 sudo -E 把环境变量带过去）

UDP 适用的环境：当网络带宽比较窄，比如 4G/3G 等移动通信系统，此时用 udp 比较合适
TCP 适用的环境：网络带宽足够，对视频花屏要求比较高，对延时要求较小的应用场景，适合 TCP

2.9 如何获取 rtsp 中原来的 timestamp

答：opencv 中默认获取的 rtsp 时间戳是从 0 开始的，如果想获取 rtsp 中的原始时间戳，可以用环境变量进行控制，然后按照问题 1 进行获取即可

```
export OPENCV_FFMPEG_CAPTURE_OPTOINS="keep_rtsp_timestamp;1"
```

注意: 外置的 options 会覆盖内部默认的设置，因此最好按照完整的 options 来设置

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="keep_rtsp_timestamp;1|buffer_size;
↪1024000|max_delay;500000|stimeout;20000000"
```

2.10 如何判断视频花屏的原因

答：这里提的视频花屏是长时间的花屏，对于偶尔的花屏有可能是网络数据传输错误导致的，此类不属于应用代码可控的方位。如果视频出现长时间的花屏，很大概率是由于视频帧读取不及时，导致内部缓存满以后，socket recv buffer 溢出导致的。

1. 将加大 rmem_max 到 2M，如果此时花屏消失，说明应用的数据处理抖动很大，应该要加大 buffer queue 进行平滑

   ```
   echo 2097152 > /proc/sys/net/core/rmem_max
   ```

2. 用 netstat -na, 一般是一下格式，找到 rtsp 的那个端口（udp 在应用中会有打印，tcp 的话可以看目标 rtsp 地址），这里的 Recv-Q, Send-Q 在正常情况应该都是 0，或者不满的，如果 Recv-Q 经常有很大的数，就说明 overflow 了。一般 Send- Q 不会出问题，如果这个也很大的话，那么很可能 network driver 驱动挂死了。

   ```
   Proto Recv-Q Send-Q Local Address Foreign Address State
   tcp 0 0 0.0.0.0:111 0.0.0.0:\* LISTEN
   ```

2.11 无法连接 rtsp ?

答：可以通过 ffmpeg 固有命令来进行连接测试：（url 为 rtsp 连接地址）

```
ffmpeg -rtsp_transport tcp -i url -f rawvideo -y /dev/null
或者
ffmpeg -rtsp_transport udp -i url -f rawvideo -y /dev/null
若以上无法连接成功，请检查网络。
```

2.12 确认解码器是否能正常工作：（url 为文件名或者 rtsp 连接地址）

答：

```
ffmpeg -i url -f rawvideo -y /dev/null
```

2.13 确认解码器和 vpp 的 OpenCV 接口是否正常工作：

答：

```
vidmulti number_of_instances url1 url2
```

2.14 解码不正确或者无法解码的最终调试手段

答：如果经常各种调试后，在现场仍然无法解决问题，可以通过打开环境变量，把问题发生前后的数据 dump 下来，供后续进行进一步分析

在 SOC 模式下

```
echo “0 0 1000 100” > /proc/vpuinfo
```

(dump 第 1 个 core 的第 1 个 instance 的码流数据)

这个配置会在两个文件之间循环存储 1000 帧数据，当问题发生的时候，把这两个发生前后的那个 1000 帧文件拷贝回来就可以。两个文件的存储位置在/data/core_%dinst%d_stream%d.bin.

2.15 判断 rtsp 是否正常工作

答:

方法一: 通过 vlc 播放视频 (推荐), 分别设置 tcp, udp 方式

方法二: 使用 vidmutil 测试程序播放, vidmutil 默认是 udp 方式, 通过设置环境变量使用 tcp 方式。

```
export OPENCV_FFMPEG_CAPTURE_OPTIONS="rtsp_transport;tcp|buffer_size;
↪1024000|max_delay;50000"
sudo -E ./vidmulti thread_num input_video [card] [enc_enable] input_video [card] [enc_enable]...
```

2.16 播放 rtsp 流出现断连情况验证

答: 可以使用 vlc 播放相同的视频, 在相同的时间下, 看 vlc 播放是否有断连的情况, 注意设置 vlc 的缓冲区大小。

2.17 验证当前 rtsp 服务输出的视频是否有花屏

答: 使用 vlc 播放视频, 持续一段时间, 看视频是否有花屏

2.18 查看 rtsp 服务是否实时推流

答: 通过 rtspserver 日志, 查看当前播放的文件是否正在发送。

2.19 对于 cvQueryFrame 等老的 opencv 接口支持状况

答:

有些客户采用旧版 opencv 的 C 接口, 接口列表如下

```
void cvReleaseCapture( CvCapture** pcapture )

IplImage* cvQueryFrame( CvCapture* capture )

int cvGrabFrame( CvCapture* capture )

IplImage\* cvRetrieveFrame( CvCapture* capture, int idx )

double cvGetCaptureProperty( CvCapture* capture, int id )

int cvSetCaptureProperty( CvCapture* capture, int id, double value )
```

(续下页)

(接上页)

```
int cvGetCaptureDomain( CvCapture* capture)

CvCapture * cvCreateCameraCapture (int index)

CvCapture * cvCreateFileCaptureWithPreference (const char * filename, int apiPreference)

CvCapture * cvCreateFileCapture (const char * filename)
```

对于这些接口，大部分都是支持的，只有返回值是 iplImage 的接口无法支持，这是因为我们硬件底层的 ion 内存类型是保存在 MAT 的 uMatData 类型中的，而 iplIMage 类型没有 uMatData 数据结构。

因此对于客户目前使用 cvQueryFrame 接口的，建议客户基于 cap.read 接口封一个返回值为 Mat 数据的 C 函数接口，不要直接调用 opencv 老版的接口。

2.20 对于 VPP 硬件不支持的 YUV 格式转换，采取什么样的软件方式最快？

答:

建议采用内部增强版本的 libyuv。

相比较原始版本，增加了许多 NEON64 优化过的格式转换 API 函数。其中包含许多 JPEG YCbCr 相关的函数。

位置：/opt/sophon/libsophon-current/lib/

2.21 OpenCV 中的 BGR 格式，在 libyuv 中对应的那个格式？OpenCV 中的 RGB 格式呢？

答:

- OpenCV 中的 BGR 格式，在 libyuv 中对应的格式为 RGB24
- OpenCV 中的 RGB 格式，在 libyuv 中对应的格式为 RAW。

Copyright © SOPHGO

2.22 若是采用 libyuv 处理 JPEG 方面的输出或者输入，需要注意什么事项？

答：

若是处理 jpeg 方面的输出或者输入，需要使用 J400，J420，J422，J444 等字样的函数，不然输出结果会有色差。

原因是 JPEG 的格式转换矩阵跟视频的不一样。

2.23 ffmpeg&opencv 支持 gb28181 协议，传入的 url 地址形式如下

答：

udp 实时流地址

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid=35018284001310090010
↪#localid=12478792871163624979#localip=172.10.18.201#localmediaport=20108
34020000002019000001:123456@35.26.240.
↪99:5666：sip服务器国标编码:sip服务器的密码@sip服务器的ip地址:sip服务器的port
deviceid：前段设备20位编码
localid：本地20位编码，可选项
localip：本地ip，可选项. 不设置会获取 eth0 的ip，如果没有eth0需要手动设置
localmediaport：媒体接收端的视频流端口，需要做端口映射，映射两个端口(rtp:11801,
↪rtcp:11802)，两个端口映射的in和out要相同.同一个核心板端口不可重复。
```

udp 回放流地址

```
gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?deviceid=\
↪35018284001310090010#devicetype=3#localid=12478792871163624979#localip=172.10.18.201
↪#localmediaport=20108#begtime=20191018160000#endtime=20191026163713
34020000002019000001:123456@35.26.240.
↪99:5666：sip服务器国标编码:sip服务器的密码@sip服务器的ip地址:sip服务器的port
deviceid：前段设备20位编码
devicetype：录像存储累类型
localid：本地20位编码，可选项. 不设置会获取 eth0 的ip，如果没有eth0需要手动设置
localip：本地ip，可选项
localmediaport：媒体接收端的视频流端口，需要做端口映射，映射两个端口(rtp:11801,
↪rtcp:11802)，两个端口映射的in和out要相同.同一个核心板端口不可重复。
begtime：录像起始时间
endtime：录像结束时间
```

tcp 实时流地址

```
gb28181://34020000002019000001:123456@35.26.240.99:5666?deviceid=35018284001310090010
↪#localid=12478792871163624979#localip=172.10.18.201
34020000002019000001:123456@35.26.240.
↪99:5666：sip服务器国标编码:sip服务器的密码@sip服务器的ip地址:sip服务器的port
deviceid：前段设备20位编码
```

(续下页)

(接上页)

```
localid：本地20位编码，可选项
localip: 本地ip，是可选项.不设置会获取 eth0 的ip，如果没有eth0需要手动设置
```

tcp 回放流地址

```
gb28181_playback://34020000002019000001:123456@35.26.240.99:5666?
↪deviceid=35018284001310090010#devicetype=3#localid=1247879287116362497 9#localip=172.10.
↪18.201#begtime=20191018160000#endtime=20191026163713
34020000002019000001:123456@35.26.240.
↪99:5666：sip服务器国标编码:sip服务器的密码@sip服务器的ip地址:sip服务器的port
deviceid：前段设备20位编码
devicetype :录像存储累类型
localid :本地20位编码，可选项
localip :本地ip，可选项. 不设置会获取 eth0 的ip，如果没有eth0需要手动设置
begtime :录像起始时间
endtime :录像结束时间
```

注意:

1. 流媒体传输默认是 udp 方式，如果使用 tcp 方式获取实时流或回放流，需要显示的指定。

   Ffmpeg 指定 tcp 方式为接口调用通过 av_dict_set 设置 gb28181_transport_rtp 为 tcp。

   Opencv 指定方式是设置环境变量

   ```
   export OPENCV_FFMPEG_CAPTURE_OPTIONS="gb28181_transport_rtp;tcp"
   ```

2. 如果使用 udp 方式外部无法访问到内部 ip/port，localmediaport 需要做端口映射，端口映射需要两个 rtp 和 rtcp。

3. 做端口映射时，使用的端口号尽量不要太大，推荐 10000–20000 的端口，socket 端口号的最大值时 65536，但是很情况下，端口号是受很多资源的限制。端口号使用过大可能会出现：[bind failed] 错误打印。

2.24 现在 opencv 中默认是使用 ION 内存作为 MAT 的 data 空间，如何指定 Mat 对象基于 system memory 内存去创建使用？

答:

```
using namespace cv;

Mat m; m.allocator = m.getDefaultAllocator();     // get system allocator
```

然后就可以正常调用各种 mat 函数了，如 m.create() m.copyto()，后面就会按照指定的 allocator 来分配内存了。

```
m.allocator = hal::getDefaultAllocator();  // get ion allocator
```

就又可以恢复使用 ION 内存分配器来分配内存。

2.25 FFMPEG JPEG 编码与转码应用示例

答:

- **调用 JPEG 编码的 ffmpeg 命令**

```
ffmpeg -c:v jpeg_bm -i src/5.jpg -c:v jpeg_bm  -is_dma_buffer 1 -y 5nx.jpg
```

- **调用动态 JPEG 转码的 ffmpeg 命令**

```
ffmpeg -c:v jpeg_bm -num_extra_framebuffers 2 -i in_mjpeg.avi -c:v jpeg_bm -is_dma_
↪buffer 1 -y test_avi.mov

ffmpeg -c:v jpeg_bm -num_extra_framebuffers 2 -i in_mjpeg.mov -c:v jpeg_bm -is_dma_
↪buffer 1 -y test_mov.mov
```

2.26 如何从 FFMPEG 的输入缓冲区中读取 bitstream?

答:

FFMPEG 源码应用示例 /opt/sophon/sophon-ffmpeg-latest/share/ffmpeg/examples/avio_reading.c (or http://www.ffmpeg.org/doxygen/trunk/doc_2examples_2avio_reading_8c-example.html)

在这一示例中，libavformat demuxer 通过 **custom AVIOContext read callback** 访问媒体信息，而不是通过在传入 FFMPEG 中的文件、rstp 等协议访问媒体信息的。

以下是 middleware-soc 中的一个使用 avio + jpeg_bm 解码静态 jpeg 图片的例子。(/opt/sophon/sophon-ffmpeg-latest/share/ffmpeg/examples/avio_decode_jpeg.c)

2.27 从内存读取图片，用 AVIOContext *avio =avio_alloc_context()，以及 avformat_open_input() 来初始化，发现初始化时间有 290ms；但是如果从本地读取图片，只有 3ms。为啥初始化时间要这么长？怎样减少初始化时间？

答:

```
ret = avformat_open_input(&fmt_ctx, NULL, NULL, NULL);
```

这里是最简单的调用。因此，avformat 内部会会读取数据，并遍历所有的数据，来确认 avio 中的数据格式。

若是避免在这个函数中读取数据、避免做这种匹配工作。在已经知道需要使用的 demuxer 的前提下，譬如，已知 jpeg 的 demuxer 是 mjpeg，可将代码改成下面的试试。

```
AVInputFormat\* input_format = av_find_input_format("mjpeg");

ret = avformat_open_input(&fmt_ctx, NULL, input_format, NULL);
```

2.28 如何查看 FFMPEG 中支持的分离器的名称?

答:

root@linaro-developer:~# ffmpeg -demuxers | grep jpeg
D jpeg_pipe piped jpeg sequence
D jpegls_pipe piped jpegls sequence
D mjpeg raw MJPEG video
D mjpeg_2000 raw MJPEG 2000 video
D mpjpeg MIME multipart JPEG
D smjpeg Loki SDL MJPEG

2.29 如何在 FFMPEG 中查看解码器信息，例如查看 jpeg_bm 解码器信息?

答:

root@linaro-developer:/home/sophgo/test# ffmpeg -decoders | grep _bm
V····· avs_bm bm AVS decoder wrapper (codec avs)
V····· cavs_bm bm CAVS decoder wrapper (codec cavs)
V····· flv1_bm bm FLV1 decoder wrapper (codec flv1)
V····· h263_bm bm H.263 decoder wrapper (codec h263)
V····· h264_bm bm H.264 decoder wrapper (codec h264)
V····· hevc_bm bm HEVC decoder wrapper (codec hevc)
V····· jpeg_bm BM JPEG DECODER (codec mjpeg)
V····· mpeg1_bm bm MPEG1 decoder wrapper (codec mpeg1video)
V····· mpeg2_bm bm MPEG2 decoder wrapper (codec mpeg2video)
V····· mpeg4_bm bm MPEG4 decoder wrapper (codec mpeg4)
V····· mpeg4v3_bm bm MPEG4V3 decoder wrapper (codec msmpeg4v3)

V····· vc1_bm bm VC1 decoder wrapper (codec vc1)
V····· vp3_bm bm VP3 decoder wrapper (codec vp3)
V····· vp8_bm bm VP8 decoder wrapper (codec vp8)
V····· wmv1_bm bm WMV1 decoder wrapper (codec wmv1)
V····· wmv2_bm bm WMV2 decoder wrapper (codec wmv2)
V····· wmv3_bm bm WMV3 decoder wrapper (codec wmv3)

2.30 如何在 FFMPEG 中查看解码器信息，例如查看 jpeg_bm 编码器信息?

答:

root@linaro-developer:/home/sophgo/test# ffmpeg -h decoder=jpeg_bm
Decoder jpeg_bm [BM JPEG DECODER]:
General capabilities: avoidprobe
Threading capabilities: none
jpeg_bm_decoder AVOptions:
-bs_buffer_size <int> .D.V..... the bitstream buffer size (Kbytes) for bm jpeg decoder (from 0 to INT_MAX) (default 5120)
-chroma_interleave <flags> .D.V..... chroma interleave of output frame for bm jpeg decoder (default 0)
-num_extra_framebuffers <int> .D.V..... the number of extra frame buffer for jpeg decoder (0 for still jpeg, at least 2 for motion jpeg) (from 0 to INT_MAX) (default 0)

2.31 如何在 FFMPEG 中查看编码器信息，例如查看 jpeg_bm 编码器信息?

答:

root@linaro-developer:/home/sophgo/test# ffmpeg -h encoder=jpeg_bm
Encoder jpeg_bm [BM JPEG ENCODER]:
General capabilities: none
Threading capabilities: none
Supported pixel formats: yuvj420p yuvj422p yuvj444p
jpeg_bm_encoder AVOptions:
-is_dma_buffer <flags> E..V..... flag to indicate if the input frame buffer is DMA buffer (default 0)

2.32 调用 API 实现 jpeg 编码的应用示例

答:

```
AVDictionary* dict = NULL;

av_dict_set_int(&dict, "is_dma_buffer", 1, 0);

ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

2.33 调用 FFMPEG 的 API 实现静态 jpeg 图片解码时设置 jpeg_bm 解码器参数的应用示例

答:

```
AVDictionary* dict = NULL;

/* bm_jpeg 解码器的输出是 chroma-interleaved模式,例如, NV12 */

av_dict_set_int(&dict, "chroma_interleave", 1, 0);

/* The bitstream buffer 为 3100KB(小于 1920x1080x3),
/* 假设最大分辨率为 1920x1080 */
av_dict_set_int(&dict, "bs_buffer_size", 3100, 0);
/* 额外的帧缓冲区: 静态jpeg设置为0, 动态mjpeg设置为2 */
av_dict_set_int(&dict, "num_extra_framebuffers", 0, 0);

ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

2.34 调用 FFMPEG 的 API 实现动态 jpeg 图片解码时设置 jpeg_bm 解码器参数的应用示例

答:

```
AVDictionary* dict = NULL;

/* bm_jpeg 解码器输出的是 chroma-separated 模式, 例如, YUVJ420 */

av_dict_set_int(&dict, "chroma_interleave", 0, 0);

/* The bitstream buffer 为 3100KB */
/* 假设最大分辨率为 1920x1080 */
av_dict_set_int(&dict, "bs_buffer_size", 3100, 0);
/* 额外的帧缓冲区: 静态jpeg设置为0, 动态mjpeg设置为2 */
av_dict_set_int(&dict, "num_extra_framebuffers", 2, 0);
```

(续下页)

(接上页)

```
ret = avcodec_open2(pCodecContext, pCodec, &dict);
```

2.35 BM168x 解码性能对于 H264/H265 有差别吗？如果调整码率的话，最多可以解多少路呢？有没有对应的数据参考？

答:

264,265 是解码路数相同的。

码率对解码帧率会有影响，这个变化就需要实测，例如我们说的 BM1686 解码能达到 960fps 是针对监控码流而言的，这类监控码流没有 B 帧，场景波动较小，码率基本在 2~4Mbps。如果是电影或者其他码率很高的，比如 10Mbps，20Mbps 甚至更多，是会有明显影响的，具体多大这个需要实测。

分辨率对于解码帧率的影响，是可以按照比例来换算的。我们说的 960fps 是针对 1920x1080 HD 分辨率而言的。

2.36 是否可以通过抽帧来提高 BM168x 的解码路数

答:

我们 opencv 中提供的抽帧，是在解码出来的结果中做的，并不是只解 I/P 帧的抽帧概念。这里的抽帧解码主要是保证出来帧数的均匀，使得后续的分析处理是等间隔的进行，这是为后续模型分析比较复杂的时候，不能达到每帧都检测而设计的解决方案，但并不能达到增加解码路数的效果。

这里顺便解释下，为什么不提供只解 I/P 帧的抽帧功能。如果只解 I、P 帧的话，抽帧的间隔就完全取决于码流的编码结构，这样是比较难控制住性能，比如监控码流中的没有 B 帧，那其实就相当于没有抽帧了。如果客户可以控制编码端，那更切合实际的做法是直接降低编码端的编码帧率，比如降到 15fps，那样解码路数就可以直接提升；反之，如果客户没有办法控制编码端，那么同样的，只解 IP 帧的抽帧方式就也无法达到增加解码路数的效果。

2.37 是否支持 avi，f4v，mov，3gp，mp4，ts，asf，flv，mkv 封装格式的 H264/H265 视频解析？

答：我们使用 ffmpeg 对这些封装格式进行支持，ffmpeg 支持的我们也支持。经查，这些封装格式 ffmpeg 都是支持的。但是封装格式对于 H265/264 的支持，取决于该封装格式的标准定义，比如 flv 标准中对 h265 就没有支持，目前国内的都是自己扩展的。

2.38 是否支持 png, jpg, bmp, jpeg 等图像格式

答: Jpg/jpeg 格式除了有 jpeg2000 外, 自身标准还有很多档次, 我们采用软硬结合的方式对其进行支持。对 jpeg baseline 的除了极少部分外, 都用硬件加速支持, 其他的 jpeg/jpg/bmp/png 采用软件加速的方式进行支持。主要的接口有 opencv/ffmpeg 库。

2.39 Valgrind 内存检查为什么有那么多警告，影响到应用的调试了

答:

我们的版本发布每次都会用 valgrind 检查一遍内存泄漏问题，如果有内存泄露问题我们会检查修正的。之所以没有去掉有些警告，是因为这些警告大部分都是内存没有初始化，如果对这些内存每个都加上初始化，会明显导致速度性能下降，而且我们确认后续操作是在硬件对其赋值后再进行的，对于此类警告，我们就不会去消除。

为了避免警告过多对上层应用调试造成影响，建议使用 valgrind 的 suppression 功能，可以通过过滤配置文件，来避免我们模块产生的 valgrind 警告，从而方便上层应用调试自身的程序。

2.40 使用 opencv 的 video write 编码，提示物理内存 (heap2) 分配失败

答:

确认 heap2 设置的大小，如果 heap2 默认大小是几十 MB，需要设置 heap2 size 为 1G。目前出厂默认配置是 1G。

```
Update_boot_info 可查询heap2 size

update_boot_info –heap2_size=0x40000000 –dev=0x0 设置heap2 size为1G。设置后重装驱动。
```

2.41 Bm_opencv 的 imread jpeg 解码结果和原生 opencv 的 imread jpeg 结果不同，有误差

答: 是的。原生 opencv 使用 libjpeg-turbo, 而 bm_opencv 采用了 bm168x 芯片中的 jpeg 硬件加速单元进行解码。

误差主要来自解码输出 YUV 转换到 BGR 的过程中。1）YUV 需要上采样到 YUV444 才能进行 BGR 转换。这个 upsample 的做法没有标准强制统一，jpeg-turbo 提供了默认 Fancy upsample，也提供了快速复制上采样的算法，原生 opencv 在 cvtColor 函数中采用了快速复制上采样算法，而在 imread 和 imdecode 中沿用了 libjpeg-turbo 默认的 fancy upsample；而 BM168x 硬件加速单元采用快速复制的算法。2）YUV444 到 BGR 的转换是浮点运算，浮点系数精度的不同会有 +/-1 的误差。其中 1）是误差的主要来源。

这个误差并不是错误，而是双方采用了不同的 upsample 算法所导致的, 即使 libjpeg-turbo 也同时提供了两种 upsample 算法。

如果用户非常关注这两者之间的差异，因为这两者之间的数值差异导致了 AI 模型精度的下降，我们建议有两种解决办法：

1）设置环境变量 export USE_SOFT_JPGDEC=1，可以指定仍然使用 libjpeg-turbo 进行解码。但是这样会导致 cpu 的 loading 上升，不推荐

2）可能过去模型太依赖开源 opencv 的解码结果了，可以用 bm_opencv 的解码结果重新训练模型，提高模型参数的适用范围。

可以使用 libjpeg-turbo 提供的 djpeg 工具对于测试工具集的数据进行重新处理，然后用处理后的数据对模型进行训练。djpeg 的命令如下：

```
./djpeg -nosmooth -bmp -outfile xxxxx.bmp input.jpg
```

然后用重新生成 bmp 文件作为训练数据集，进行训练即可。

2.42 如何查看 vpu/jpu 的内存、使用率等状态

答：

在 soc 模式下，可以用下面的方法查看：

```
cat /proc/vpuinfo

cat /proc/jpuinfo
```

2.43 视频支持 32 路甚至更多的时候，报视频内存不够使用，如何优化内存使用空间

答：

在 soc 模式下，视频内存的默认配置是 2G，正常使用在 16 路是绰绰有余的，但在 32 路视频需要在应用层面上仔细设计，不能有任何的浪费。

如果解码使用的是 FFMPEG 框架，首先保证视频输出格式使用压缩格式，即 output_format 101。Opencv 框架的话，内部已经默认使用压缩格式了；

其次如果应用在获取到解码输出 avFrame 后，并不是直接压入队列，而是转换到 RGB 或者非压缩数据后再缓存的话，可以用 av_dict_set extra_frame_buffer_num 为 1（默认为 2）。Opencv 内部在最新版本中会默认优化。

最后，如果以上优化过后，仍然不够的话，在 soc 模式下可以考虑更改 dtb 设置，给视频挪用分配更多的内存，当然相应的，其他模块就要相应的减少内存。这个要从系统角度去调配。

2.44 Opencv 中 mat 是如何分配设备内存和系统内存的？

答：

因为受设计影响，这个问题细节比较多，主要从三方面能解释。

1. 在 soc 模式下，设备内存和系统内存是同一份物理内存，通过操作系统的 ION 内存进行管理，系统内存是 ION 内存的 mmap 获得。

1. 在 sophon opencv 中默认会同时开辟设备内存和系统内存，其中系统内存放在 mat.u->data 或 mat.data 中，设备内存放在 mat.u->addr 中。只有以下几种情况会不开辟设备内存，而仅提供系统内存：

   · 当 data 由外部开辟并提供给 mat 的时候。即用以下方式声明的时候：

   ```
   Mat mat(h, w, type, data); 或 mat.create(h, w, type, data);
   ```

   · 在 soc 模式下，当 type 不属于（CV_8UC3, CV_32FC1, CV_32FC3）其中之一的时候。这里特别注意 CV_8UC1 是不开辟的，这是为了保证我们的 opencv 能够通过开源 opencv 的 opencv_test_core 的一致性验证检查。

   · 当宽或者高小于 16 的时候。因为这类宽高，硬件不支持

2. 在 BM1686 的 SOC 模式下，mat 分配的 CV_8UC3 类型的设备内存会自动做 64 对齐，即分配的内存大小一定是 64 对齐的（注意：仅对 soc 模式的 CV_8UC3 而言，且仅对 BM1686 芯片）。

2.45 ffmpeg 中做图像格式/大小变换导致视频播放时回退或者顺序不对的情况处理办法

答：ffmpeg 在编码的时候底层维护了一个 avframe 的队列作为编码器的输入源，编码期间应保证队列中数据有效，如果在解码后需要缩放或者像素格式转换时候需要注意送进编码器的 avframe 的数据有效和释放问题。

在例子 ff_bmcv_transcode 中从解码输出 src-avframe 转换成 src-bm_image 然后做像素格式转换或者缩放为 dst-bm_image 最后转回 dst-avframe 去编码的过程中 src-avframe、src-bm_image 的设备内存是同一块，dst- avframe、dst-bm_image 的设备内存是同一块。在得到 dst-bm_image 后即可释放 src_avframe 和 src-bm_image 的内存（二者释放其中一个即可释放设备内存），作为编码器的输入 dts-bm_image 在转换成 dst- avframe 之后其设备内存依然不能被释放（常见的异常情况是函数结束 dts-bm_image 的引用计数为 0 导致其被释放），如果 dst-bm_image 被释放了此时用 dst-avframe 去编码结果肯定会有问题。

解决方法是 dst-bm_image 的指针是 malloc 一块内存，然后将其传给 av_buffer_create，这样就保证在函数结束的时候 dst-bm_image 引用计数不会减 1，释放的方法是将 malloc 的 dst-bm_image 指针通过 av_buffer_create 传给释放的回调函数，当 dst- avframe 引用计数为 0 的时候会调用回调函数将 malloc 的指针和 dst-bm_image 的设备内存一起释放。详见例子 ff_bmcv_transcode/ff_avframe_convert.cpp。

2.46 启动设备首次执行某个函数慢，重启进程再次运行正常

现象：设备上电后第一次执行程序，函数处理时间长，再次执行程序，运行正常。

解决：先做个验证，如果不重启可复现，就说明是文件 cache 导致的变慢。

1. 上电后第一次执行慢，第二次执行正常，之后进入 root 用户
2. 清除 cache echo 3 > /proc/sys/vm/drop_caches
3. 再次执行程序，运行慢，即可确定是 cache 导致的。

2.47 Opencv mat 创建失败，提示"terminate called after throwing an instance of 'cv::Exception' what(): OpenCV(4.1.0) ··· ···matrix.cpp:452: error: (-215:Assertion failed) u != 0 in function 'creat'"

答：

这种错误主要是设备内存分配失败。失败的原因有两种：

1. 句柄数超过系统限制，原因有可能是因为句柄泄漏，或者系统句柄数设置过小，可以用如下方法确认：

   查看系统定义的最大句柄数：

   ```
   ulimit -n
   ```

   查看当前进程所使用的句柄数：

   ```
   lsof -n|awk '{print $2}'|sort|uniq -c|sort -nr|more
   ```

2. 设备内存不够用。可以用如下方法确认：

   · SOC 模式下

   ```
   cat /sys/kernel/debug/ion/bm_vpp_heap_dump/summary
   ```

解决方案：在排除代码本身的内存泄漏或者句柄泄漏问题后，可以通过加大系统最大句柄数来解决句柄的限制问题：ulimit -HSn 65536

设备内存不够就需要通过优化程序来减少对设备内存的占用，或者通过修改 dts 文件中的内存布局来增加对应的设备内存。详细可以参考 SM5 用户手册中的说明。

2.48 opencv 转 bm_image 的时候，报错 "Memory allocated by user, no device memory assigned. Not support BMCV!"

答：这种错误通常发生在 soc 模式下，所转换的 Mat 只分配了系统内存，没有分配设备内存，而 bm_image 要求必须有设备内存，因此转换失败。

会产生这类问题的 Mat 通常是由外部分配的 data 内存 attach 过去的，即调用 Mat(h, w, data) 或者 Mat.create(h,w, data) 来创建的，而 data!=NULL, 由外部分配。

对于这种情况，因为 bm_image 必须要求设备内存，因此解决方案有

1. 新生成个 Mat，默认创建设备内存，然后用 copyTo() 拷贝一次，把数据移到设备内存上，再重新用这个 Mat 来转成 bm_image
2. 直接创建 bm_image，然后用 bm_image_copy_host_to_device, 将 Mat.data 中的数据拷贝到 bm_image 的设备内存中。

2.49 Opencv 用已有 Mat 的内存 data，宽高去创建新的 Mat 后，新 Mat 保存的图像数据错行，显示不正常

答：保存的图像错行，通常是由于 Mat 中 step 信息丢失所造成。

一般用已有 Mat 去生成一个新 Mat，并且要求内存复用时，可以直接赋值给新的 Mat 来简单实现，如 Mat1 = Mat2.

但在某些情况下，比如有些客户受限于架构，函数参数只能用 C 风格的指针传递，就只能用 Mat 中的 data 指针，rows，cols 成员来重新恢复这个 Mat。这时候就需要注意 step 变量的设置，在默认情况下是 AUTO_STEP 配置，即每行数据没有填充数据。但是在很多种情况下，经过 opencv 处理后，会导致每行出现填充数据。如，

1. soc 模式下，我们的 Mat 考虑执行效率，在创建 Mat 内存时每行数据会做 64 字节对齐，以适配硬件加速的需求（仅在 soc 模式下）
2. opencv 的固有操作，如这个 Mat 是另一个 Mat 的子矩阵（即 rect 的选定区域），或者其他可能导致填充的操作。

因此，按照 opencv 定义，通用处理方式就是在生成新的 Mat 的时候必须指定 step，如下所示:

```
cv::Mat image_mat = cv::imread(filename,IMREAD_COLOR,0);
cv::Mat image_mat_temp(image_mat.rows,image_mat.cols,CV_8UC3,image_mat.data,image_
↪mat.step[0]);
cv::imwrite("sophgo1.jpg",image_mat_temp);
```

2.50 在 soc 模式下客户用 ffmpeg 解码时拿到 AVframe 将 data[0-3] copy 到系统内存发现 copy 时间是在 20ms 左右而相同数据量在系统内存两块地址 copy 只需要 1-3ms

答：上述问题的原因是系统在 ffmpeg 中默认是禁止 cache 的，因此用 cpu copy 性能很低，使能 cache 就能达到系统内存互相 copy 同样的速度。可以用以下接口使能 cache.

```
av_dict_set_int(&opts, "enable_cache", 1, 0);
```

但是这样直接 copy 数据保存是非常占用内存、带宽和 cpu 算力的，我们推荐采用零拷贝的方式来实现原始解码数据的保存：

1. 推荐使用 extra_frame_buffer_num 参数指定增大硬件帧缓存数量，可以根据自己的需要选择缓存帧的数量。这个方式的弊端，一个是占用解码器内存，可能减少视频解码的路数；另一个是当不及时释放，当缓存帧全部用完时，会造成视频硬件解码堵塞。

   ```
   av_dict_set_int(&opts, "extra_frame_buffer_num", extra_frame_buffer_num, 0);
   ```

2. 推荐使用 output_format 参数设置解码器输出压缩格式数据，然后使用 vpp 处理输出非压缩 yuv 数据（当需要缩放，crop 时，可以同步完成），然后直接零拷贝引用非压缩 yuv 数据。这种方式不会影响到硬件解码性能，并且可以缓存的数据空间也大很多。

   ```
   av_dict_set_int(&opts, "output_format", 101, 0);
   ```

2.51 在 opencv VideoCapture 解码视频时提示: maybe grab ends normally, retry count = 513

上述问题是因为在 VideoCapture 存在超时检测，如果在一定时间未收到有效的 packet 则会输出以上 log，此时如果视频源是网络码流可以用 vlc 拉流验证码流是否正常，如果是文件一般是文件播放到末尾需调用 VidoeCapture.release 后重新 VideoCapture.open

2.52 [问题分析] 客户反馈碰到如下错误提示信息”VPU_DecRegisterFrameBuffer failed Error code is 0x3”，然后提示 Allocate Frame Buffer 内存失败。

这个提示信息表示：分配的解码器缓存帧个数，超过了最大允许的解码帧。导致这个问题的原因有可能是：

1. 不支持的视频编码格式，比如场格式，此时可以用 FAQ14 的方法，把码流数据录下来，提交给我们分析。
2. 设置了过大的 extra_frame_buffer_num。理论上，extra_frame_buffer_num 不能超过 15，超过了以后就有可能不能满足标准所需的最大缓存帧数。因为

大部分编码码流并没有用到最大值，所以 extra_frame_buffer_num 大于 15 的时候，对大部分码流仍然是可以工作的。

目前发现可能导致这个问题的原因有上述两种，后续有新的案例继续增补

2.53 SOC 模式下，opencv 在使用 8UC1 Mat 的时候报错，而当 Mat 格式为 8UC3 的时候，同样的程序完全工作正常。

这个问题碰到的客户比较多，这次专门设立一个 FAQ 以便搜索。其核心内容在 FAQ46 “Opencv 中 mat 是如何分配设备内存和系统内存的”有过介绍，可以继续参考 FAQ46

在 soc 模式下，默认创建的 8UC1 Mat 是不分配设备内存的。因此当需要用到硬件加速的时候，比如推理，bmcv 操作等，就会导致各种内存异常错误。

解决方案可以参看 FAQ26 “如何指定 Mat 对象基于 system memory 内存去创建使用”，指定 8UC1 Mat 在创建的时候，内部使用 ion 分配器去分配内存。如下所示。

```
cv::Mat gray_mat;
gray_mat.allocator = hal::getAllocator();
gray_mat.create(h, w, CV_8UC1);
```

2.54 调用 bmcv_image_vpp_convert_padding 接口时，报缩放比例超过 32 倍的错：“vpp not support: scaling ratio greater than 32”。

bm1686 的 vpp 中硬件限制图片的缩放不能超过 32 倍（bm1686 的 vpp 中硬件限制图片的缩放不能超过 128 倍）。即应满足 dst_crop_w <= src_crop_w * 32，src_crop_w <= dst_crop_w * 32，dst_crop_h <= src_crop_h * 32 ，src_crop_h <= dst_crop_h * 32。

此问题原因可能是：

1. 输入 crop_rect 中的 crop_w，crop_h 与输出 padding_attr 中的 dst_crop_w ，dst_crop_h 缩放比例超过了 32 倍。
2. crop_rect，padding_attr 值的数量应与 output_num 的数量一致。

2.55 [问题分析] 程序提示 “VPU_DecGetOutputInfo decode fail framdIdx xxx error(0x00000000) reason(0x00400000), reasonExt(0x00000000)” 是可能什么问题，这里 reason 的具体数值可能不同

这个提示通常是由码流错误造成的，提示的含义是第 xxx 帧解码错误，错误原因为….。这里具体原因对于上层应用来说，不用关心，只需知道这是由码流错误导致的即可。

进一步分析，导致码流错误的原因通常可以分为两类，我们要有针对的进行处理。因为一旦频繁出现这种提示，说明解码出来的数据是不正确的，这时候有可能是各种马赛克或者图像花，对于后续的处理会造成各种异常情况，所以我们必须尽量减少这种情况的发生。

1. 网络情况导致的丢包。这时候可以用我们的测试程序 vidmulti 验证下，如果 vidmulti 没有解码错误，那么可以排除这种情况。如果确认网络丢包的话，要分辨下是否网络带宽本身就不够，如果本身带宽不够，那没有办法，只能降低视频码流的码率。如果带宽是够的，要检查下网线。当码流连接数超过 20 多路的时候，这时候有可能已经超出百兆了，这时网线也必须换到 CAT6，与千兆网相匹配
2. 解码性能达到上限造成丢包。这种情况发生在流媒体环境中，对于文件播放是不会发生的。这时也可以用我们的 vidmulti 跑一下，作为比较。如果 vidmulti 也发生错误，说明性能确实到了上限了，否则说明应用本身还有优化的空间。

2.56 [问题分析] 程序提示“coreIdx 0 InstIdx 0: VPU interrupt wait timeout”，这是怎么回事?

这个提示表示视频解码或者编码中断超时。这个提示只是警告，会再次尝试，因此只要没有连续出现就可以忽略。这种情况一般是由解码数据错误导致或者负荷过重产生的。例如在板卡情况下，由于板卡数据交换过于频繁，造成解码或者编码数据传输堵塞，使得中断超时。

2.57 采用 TCP 传输码流的时候如果码流服务器停止推流，ffmpeg 阻塞在 av_read_frame

这是因为超时时间过长导致的，可以用一下参数设置超时时间减短。

av_dict_set(&options, “stimeout”, “1000000”, 0);

2.58 [问题分析] 当用 ffmpeg jpeg_bm 解码超大 JPEG 图片的时候，有时候会报 “ERROR:DMA buffer size(5242880) is less than input data size(xxxxxxx)”，如何解决？

在用 FFMPEG 的 jpeg_bm 硬件解码器解码 JPEG 图片的时候，默认的输入 buffer 是 5120K。在拿到 JPEG 文件前提前分配好输入缓存，在 MJPG 文件解码时可以避免频繁地创建和销毁内存。当出现默认输入 buffer 大小比输入 jpeg 文件小的时候，可以通过下面的命令来调大输入缓存。

av_dict_set_int(&opts, “bs_buffer_size” , 8192, 0); //注意：bs_buffer_size 是以 Kbyte 为单位的

2.59 调用 bmcv_image_vpp_basic 接口时，csc_type_t，csc_type 和 csc_matrix_t* matrix 该如何填？

bmcv 中 vpp 在做 csc 色彩转换时，默认提供了 4 组 601 系数和 4 组 709 系数，如 csc_type_t 所示。

1. csc_type 可以填为 CSC_MAX_ENUM，matrix 填 NULL，会默认配置 601 YCbCr 与 RGB 互转系数。
2. csc_type 可以填 csc_type_t 中参数，如 YCbCr2RGB_BT709，matrix 填 NULL，会按照所选类型配置对应系数。
3. csc_type 可以填 CSC_USER_DEFINED_MATRIX，matrix 填自定义系数。会按照自定义系数配置。

csc_matrix_t 中系数参考如下公式：

Y = csc_coe00 * R + csc_coe01 * G + csc_coe02 * B + csc_add0;

U = csc_coe10 * R + csc_coe11 * G + csc_coe12 * B + csc_add1;

V = csc_coe20 * R + csc_coe21 * G + csc_coe22 * B + csc_add2;

由于 bm1686 vpp 精度为 FP32，整数处理。

csc_coe 与 csc_add 的计算方法为: csc_coe = round（浮点数 * 1024）后按整数取补码。

csc_coe 取低 13bit，即 csc_coe = csc_coe & 0x1fff，csc_add 取低 21bit，即 csc_add = csc_add & 0x1fffff。

举例如下：

floating-point coe matrix => fixed-point coe matrix

0.1826 0.6142 0.0620 16.0000 => 0x00bb 0x0275 0x003f 0x004000

2.60 [问题分析] 不同线程对同一个 bm_imag 调用 bm_image_destroy 时，程序崩溃。

bm_image_destroy(bm_image image) 接口设计时，采用了结构体做形参，内部释放了 image.image_private 指向的内存，但是对指针 image.image_private 的修改无法传到函数外，导致第二次调用时出现了野指针问题。

为了使客户代码对于 sdk 的兼容性达到最好，目前不对接口做修改。建议使用 bm_image_destroy（image）后将 image.image_private = NULL，避免多线程时引发野指针问题。

2.61 如何跨进程传递 Mat 信息，使不同进程间零拷贝地共享 Mat 中的设备内存数据？

跨进程共享 Mat 的障碍在于虚拟内存和句柄在进程间共享非常困难，因此解决这个问题的本质是：如何由一块设备内存，零拷贝地重构出相同的 Mat 数据结构。

解决这个问题会用到下面三个接口，其中前两个接口用于重构 yuvMat 的数据，后一个接口用于重构 opencv 标准 Mat 的数据。

```
cv::av::create(int height, int width, int color_format, void *data, long addr, int fd, int* plane_
↪stride, int* plane_size, int color_space = AVCOL_SPC_BT709, int color_range = AVCOL_
↪RANGE_MMPEG, int id = 0)
cv::Mat(AVFrame *frame, int id)
Mat::create(int height, int width, int total, int _type, const size_t* _steps, void* _data,unsigned
↪long addr, int fd, int id = 0)
```

```
/* 完整的跨进程共享Mat的代码如下所示。跨进程共享的方法很多，下面的例子目的在于展示如何
  使用上面的函数重构Mat数据，其他的代码仅供参考。其中image为需要被共享的Mat */
   union ipc_mat{
     struct{
         unsigned long long addr;
         int total;
         int type;
         size_t step[2];
         int plane_size[4];
         int plane_step[4];
         int pix_fmt;
         int height;
         int width;
         int color_space;
         int color_range;
         int dev_id;
         int isYuvMat;
     }message;
     unsigned char data[128];
  }signal;
```

（续下页）

(接上页)

```
memset(signal.data, 0, sizeof(signal));

if (isSender){  // 后面是send的代码
    int fd = open("./ipc_sample", O_WRONLY);
    signal.message.addr = image.u->addr;
    signal.message.height = image.rows;
    signal.message.width = image.cols;
    signal.message.isYuvMat = image.avOK() ? 1 : 0;
    if (signal.message.isYuvMat){  // 处理yuvMat
                        //avAddr(4~6)对应设备内存
        signal.message.plane_size[0] = image.avAddr(5) - image.avAddr(4);
        signal.message.plane_step[0] = image.avStep(4);

        signal.message.pix_fmt = image.avFormat();
        if (signal.message.pix_fmt == AV_PIX_FMT_YUV420P){
            signal.message.plane_size[2] =
            signal.message.plane_size[1] = image.avAddr(6) - image.avAddr(5);
            signal.message.plane_step[1] = image.avStep(5);
            signal.message.plane_step[2] = image.avStep(6);
        } else if (signal.message.pix_fmt == AV_PIX_FMT_NV12){
            signal.message.plane_size[1] = signal.message.plane_size[0] / 2;
            signal.message.plane_step[1] = image.avStep(5);
        }  // 此处仅供展示，更多的色彩格式可以继续扩展

        signal.message.color_space = image.u->frame->colorspace;
        signal.message.color_range = image.u->frame->color_range;

        signal.message.dev_id = image.card;
    } else { // 处理bgrMat
        signal.message.total = image.total();
        signal.message.type = image.type();
        signal.message.step[0] = image.step[0];
        signal.message.step[1] = image.step[1];
    }

    write(fd, signal.data, 128);

    while(1) sleep(1); //此处while(1)仅供举例，要注意实际应用中后面还需要close(fd)
} else if (!isSender){
    if ((mkfifo("./ipc_example", 0600) == -1) && errno != EEXIST){  // ipc共享仅供举例
        printf("mkfifo failed\n");
        perror("reason");
    }

    int fd = open("./ipc", O_RDONLY);
    Mat f_mat;  // 要重构的共享Mat
    int cnt = 0;

    while (cnt < 128){
        cnt += read(fd, signal.data+cnt, 128-cnt);
    }
```

(续下页)

(接上页)

```
    if(signal.message.isYuvMat) { // yuvMat
        AVFrame *f = cv::av::create(signal.message.height,
                                    signal.message.width,
                                    signal.message.pix_fmt,
                                    NULL,
                                    signal.message.addr,
                    /* 这里fd直接给0即可，其作用仅表示存在外部给的设备内存地址addr */
                                    0,
                                    signal.message.plane_step,
                                    signal.message.plane_size,
                                    signal.message.color_space,
                                    signal.message.color_range,
                                    signal.message.dev_id);
        f_mat.create(f, signal.message.dev_id);
    } else {
        f_mat.create(signal.message.height,
                     signal.message.width,
                     signal.message.total,
                     signal.message.type,
                     signal.message.step,
                     NULL,
                     signal.message.addr,
                    /* 这里fd直接给0即可，其作用仅表示存在外部给的设备内存地址addr */
                    0,
                     signal.message.dev_id);
        bmcv::downloadMat(f_mat);
                    /* 注意这里需要将设备内存的数据及时同步到系统内存中。因为yuvMat使
                     用中约定设备内存数据永远是最新的，bgrMat使用中约定系统内存数据永远
                     是最新的，这是我们opencv中遵循的设计原则 */
    }
    close(fd);
}

/* 以上代码仅供参考，请使用者根据自己实际需要修改定制 */
```

2.62 申请设备内存失败，错误返回-24。

设备内存每一次申请都会有一个 fd，ubuntu 上最大 1024。如果持续申请且不释放，fd 数量超过 1024，就会导致申请设备内存失败，错误返回-24。如果想扩大 ubuntu 的 fd 数量，可通过 ulimit 命令修改限制。如 ulimit -n 10000 可将 ubuntu 的 fd 数量扩大至 10000。

2.63 bm_image_create、bm_image_alloc_dev_mem、bm_image_attach 相关疑问。

1. bm_image_create：用于创建 bm_image 结构体。
2. bm_image_alloc_dev_mem：申请设备内存，且内部会 attach 上 bm_image。
3. bm_image_attach：用于将从 opencv 等处获取到的设备内存与 bm_image_create 申请到的 bm_image 进行绑定。

2.64 bm_image_destroy、bm_image_detach 相关疑问。

1. bm_image_detach：用于将 bm_image 关联的设备内存进行解绑，如果设备内存是内部自动申请的，才会释放这块设备内存；如果 bm_image 未绑定设备内存，则直接返回。
2. bm_image_destroy：该函数内部嵌套调用 bm_image_detach。也就是说，调用该函数，如果 bm_image 绑定的设备内存是通过 bm_image_alloc_dev_mem 申请，就会释放；如果是通过 bm_image_attach 绑定的设备内存则不会被释放，此时需要注意该设备内存是否存在内存泄露，如果存在其他模块继续使用，则由相应模块进行释放。
3. 总的来说，设备内存由谁申请则由谁释放，如果是通过 attach 绑定的设备内存，则不能调用 bm_image_destroy 进行释放，如果确定 attach 绑定的设备内存不再使用，可通过 bm_free_device 等接口进行释放。

2.65 在 bmcv::toBMI 之前是否需要调用 bm_create_image，如果调用，在最后使用 bm_image_destroy 会不会引起内存泄露？

1. bmcv::toBMI 内部嵌套调用 bm_image_create，无需再次调用 bm_create_image。
2. 如果在 bmcv::toBMI 前调用了 bm_create_image，会导致内存泄露。
3. 调用 bmcv::toBMI 后，除了需要调用 bm_image_destroy，还需要 image.image_private = NULL。